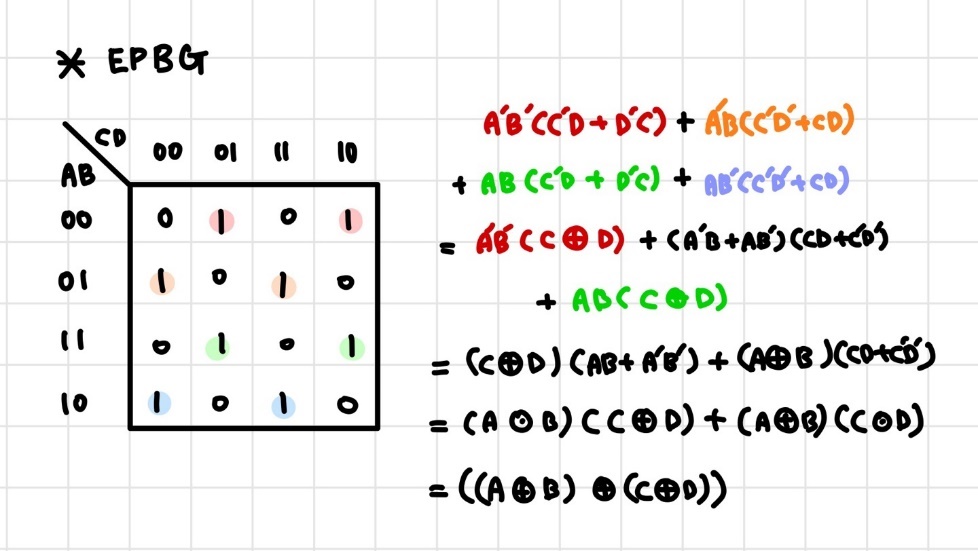
7주차 결과보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20191599 이름: 송경호

**1.**

**1-1) Even Parity Bit Generator (EPBG)**

****

Even Parity Bit Generator (이하 EPBG)의 Boolean function을 구하기 위하여 K-map을 생성하였다. Even parity bit은 다 입력값의 1의 개수가 홀수인 경우, 전체 1의 개수를 짝수로 맞춰주기 위해 1이된다. 위 K-map에서 볼 수 있듯이, ABCD가 0001, 0100, 1011 등 1의 개수가 홀수인 경우 Parity bit이 1이 되는 것을 알 수 있다.

K-map을 바탕으로 SOP form의 Boolean function을 구했을 때 even parity bit은 결과적으로 아래와 같은 Boolean function으로 나타난다. 자세한 유도 과정은 K-map옆에 명시했으며 이때 A’B + B’A = A ⊕ B이며 AB + A’B’ = (A ⊕ B)’ = A ⨀ B임을 이용했다.

**(A ⊕ B ⊕ C ⊕ D)**

구한 Boolean function을 이용하여 다음과 같이 Verilog 코드를 작성했다.

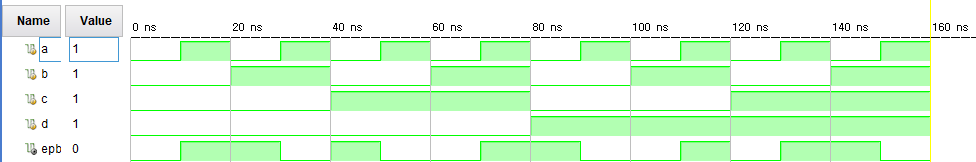
텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

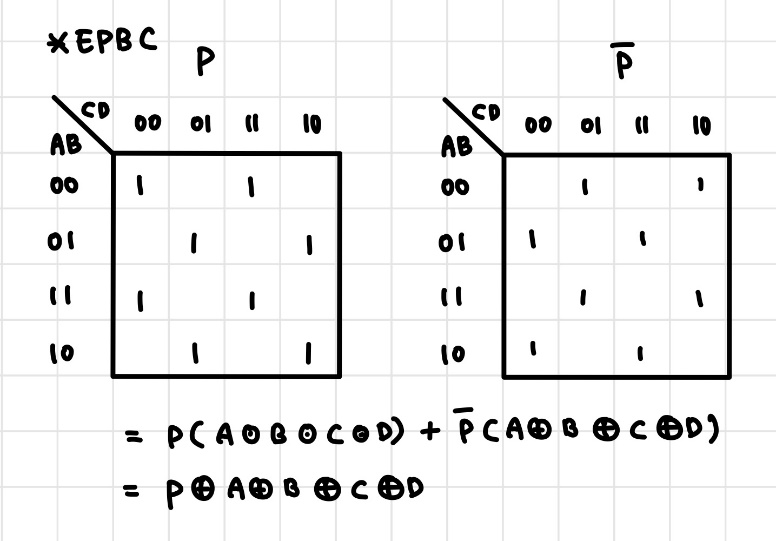
even parity bit는 4개의 입력을 XOR 연산한 결과임을 K-map을 통해 알았으므로 이를 디자인 코드에 작성했다. XOR연산은 ‘^’을 통해 구현했다. 또한 총 4개의 입력이 각 각 10,20,40,80ms마다 전환되도록 테스트 밴치코드를 구현하여 총 160ms 내에 모든 입력 경우의 수를 확인할 수 있다. 시뮬레이션 결과와 진리표는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **Out E** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |



앞서 말했듯이, 4개의 입력에서 1의 개수가 홀수일 경우 even parity bit이 1로 set된다. 시뮬레이션 결과와 진리표 모두 이러한 결과를 보여주며 두 결과값이 동일하다는 점을 확인할 수 있다.

**1-2) Even Parity Bit Checker (EPBC)**

****

Even Parity Bit Checker (이하 EPBC)의 Boolean function을 구하기 위하여 K-map을 생성하였다. Even parity checker bit은 입력 값을 바탕으로 오류를 확인하는 bit이므로 1의 경우 오류가 발생했음을 의미한다. 즉, 1의 총 개수가 짝수일 경우 오류가 존재하지 않는 것이므로 입력값이 ABCD일 때 만약 Parity bit이 1일 경우 ABCD의 1의 개수가 짝수인 경우, Parity bit가 0일 경우 ABCD의 1의 개수가 홀수인 경우일 때 오류가 발생한 것이다. 예를 들어 Parity bit가 1인 좌측 K-map을 봤을 때, ABCD가 0000, 0011 같이 1의 개수가 짝수인 경우 Parity bit 까지 합쳐져서 1의 개수가 홀수가 되어 오류가 발생한 것이다. 따라서 checker bit가 1이된다.

K-map을 바탕으로 SOP form의 Boolean function을 구했을 때 even parity check bit은 결과적으로 아래와 같은 Boolean function으로 나타난다. 자세한 유도 과정은 K-map옆에 명시했으며 이때 A’B + B’A = A ⊕ B이며 AB + A’B’ = (A ⊕ B)’ = A ⨀ B임을 이용했다.

**P ⊕ (A ⊕ B ⊕ C ⊕ D)**

구한 Boolean function을 이용하여 다음과 같이 Verilog 코드를 작성했다.

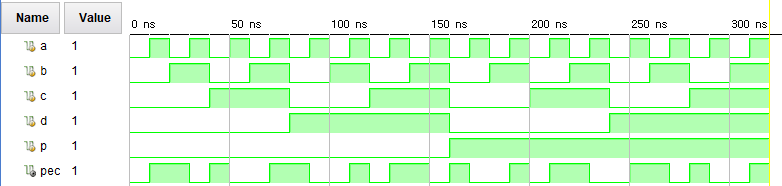
텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

even parity check bit는 5개의 입력을 XOR 연산한 결과임을 K-map을 통해 알았으므로 이를 디자인 코드에 작성했다. XOR연산은 ‘^’을 통해 구현했다. 또한 총 5개의 입력이 각 각 10,20,40,80,160ms마다 전환되도록 테스트 밴치코드를 구현하여 총 320ms 내에 모든 입력 경우의 수를 확인할 수 있다. 시뮬레이션 결과와 진리표는 아래와 같다.

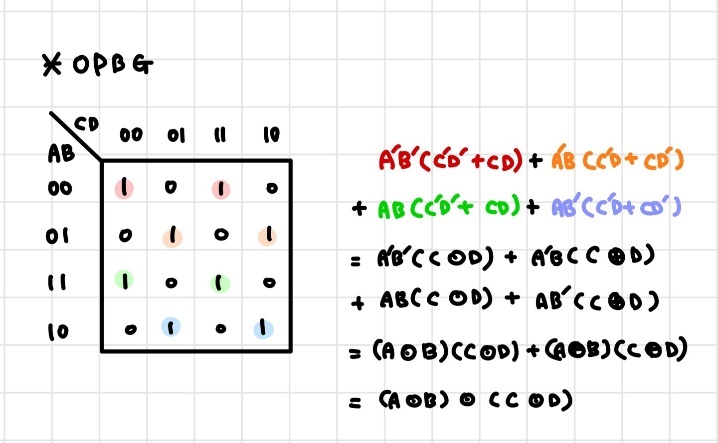
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parity Bit** | **In A** | **In B** | **In C** | **In D** | **PEC** |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |



앞서 말했듯이, 5개의 입력에서 1의 개수가 홀수일 경우 even parity check bit이 1로 set된다. 시뮬레이션 결과와 진리표 모두 이러한 결과를 보여주며 두 결과값이 동일하다는 점을 확인할 수 있다.

**2.**

**2-1) Odd Parity Bit Generator (OPBG)**

****

Odd Parity Bit Generator (이하 OPBG)의 Boolean function을 구하기 위하여 K-map을 생성하였다. Odd parity bit은 다 입력값의 1의 개수가 짝수인 경우, 전체 1의 개수를 홀수로 맞춰주기 위해 1이된다. 위 K-map에서 볼 수 있듯이, ABCD가 0000, 0101, 1010 등 1의 개수가 짝수인 경우 Parity bit이 1이 되는 것을 알 수 있다.

K-map을 바탕으로 SOP form의 Boolean function을 구했을 때 odd parity bit은 결과적으로 아래와 같은 Boolean function으로 나타난다. 자세한 유도 과정은 K-map옆에 명시했으며 이때 A’B + B’A = A ⊕ B이며 AB + A’B’ = (A ⊕ B)’ = A ⨀ B임을 이용했다.

**(A** ⨀ **B** ⨀ **C** ⨀ **D) = (A ⊕ B ⊕ C ⊕ D)’**

구한 Boolean function을 이용하여 다음과 같이 Verilog 코드를 작성했다.

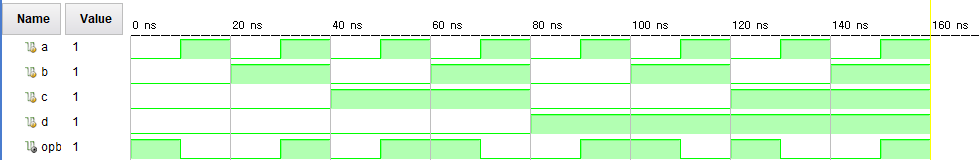
텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

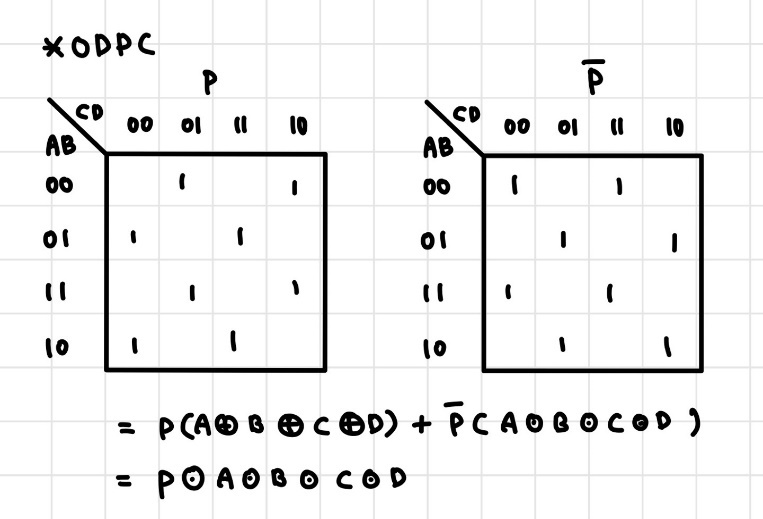
odd parity bit는 4개의 입력을 XOR 연산한 결과에 NOT연산을 한 것임을 K-map을 통해 알았으므로 이를 디자인 코드에 작성했다. XOR연산은 ‘^’을 통해 구현했다. 또한 총 4개의 입력이 각 각 10,20,40,80ms마다 전환되도록 테스트 밴치코드를 구현하여 총 160ms 내에 모든 입력 경우의 수를 확인할 수 있다. 시뮬레이션 결과와 진리표는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **Out E** |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |



앞서 말했듯이, 4개의 입력에서 1의 개수가 짝수일 경우 odd parity bit이 1로 set된다. 시뮬레이션 결과와 진리표 모두 이러한 결과를 보여주며 두 결과값이 동일하다는 점을 확인할 수 있다.

**2-2) Odd Parity Bit Checker (OPBC)**



Odd Parity Bit Checker (이하 OPBC)의 Boolean function을 구하기 위하여 K-map을 생성하였다. Odd parity checker bit은 입력 값을 바탕으로 오류를 확인하는 bit이므로 1의 경우 오류가 발생했음을 의미한다. 즉, 1의 총 개수가 홀수일 경우 오류가 존재하지 않는 것이므로 입력값이 ABCD일 때 만약 Parity bit이 1일 경우 ABCD의 1의 개수가 홀수인 경우, Parity bit가 0일 경우 ABCD의 1의 개수가 짝수인 경우일 때 오류가 발생한 것이다. 예를 들어 Parity bit가 1인 좌측 K-map을 봤을 때, ABCD가 0001, 1011 같이 1의 개수가 홀수인 경우 Parity bit 까지 합쳐져서 1의 개수가 짝수가 되어 오류가 발생한 것이다. 따라서 checker bit가 1이된다.

K-map을 바탕으로 SOP form의 Boolean function을 구했을 때 odd parity check bit은 결과적으로 아래와 같은 Boolean function으로 나타난다. 자세한 유도 과정은 K-map옆에 명시했으며 이때 A’B + B’A = A ⊕ B이며 AB + A’B’ = (A ⊕ B)’ = A ⨀ B임을 이용했다.

**P** ⨀ **A** ⨀ **B** ⨀ **C** ⨀ **D = (P ⊕ A ⊕ B ⊕ C ⊕ D)’**

구한 Boolean function을 이용하여 다음과 같이 Verilog 코드를 작성했다.

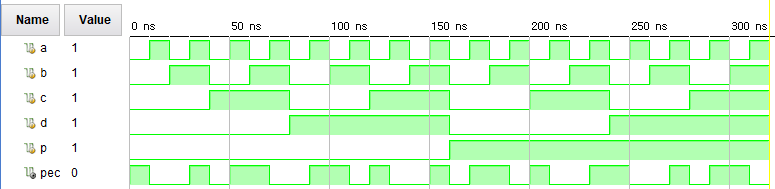
텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

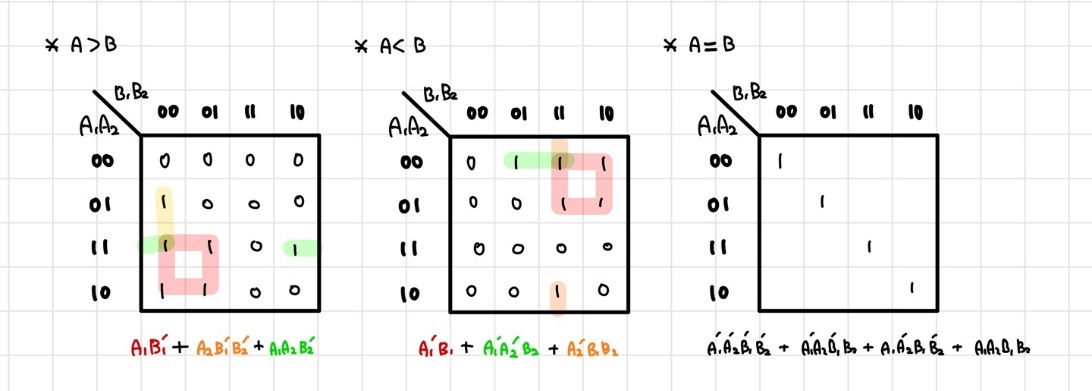
odd parity check bit는 5개의 입력을 XOR 연산한 결과에 NOT 연산을 한것임을 K-map을 통해 알았으므로 이를 디자인 코드에 작성했다. XOR연산은 ‘^’을 통해 구현했다. 또한 총 5개의 입력이 각 각 10,20,40,80,160ms마다 전환되도록 테스트 밴치코드를 구현하여 총 320ms 내에 모든 입력 경우의 수를 확인할 수 있다. 시뮬레이션 결과와 진리표는 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parity Bit** | **In A** | **In B** | **In C** | **In D** | **PEC** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |



앞서 말했듯이, 5개의 입력에서 1의 개수가 짝수일 경우 odd parity check bit이 1로 set된다. 시뮬레이션 결과와 진리표 모두 이러한 결과를 보여주며 두 결과값이 동일하다는 점을 확인할 수 있다.

**3.**

****

2 bit binary comparator는 2 bit 입력값 A, B에 대해 A와 B간의 대소 관계를 확인한다. A와 B간의 대소관계는 총 3가지로 A > B, A < B 그리고 A = B인 경우가 발생할 수 있다. 각 경우에 대한 K-map은 위와 같다.

K-map을 바탕으로 각 경우에 대한 SOP form의 Boolean function을 구했을 때 아래와 같은 Boolean function으로 나타난다.

1. A > B – A1B1‘ + A2B1‘B2’ + A1A2B2’

2. A < B - A1’B1 + A1’A2‘B2 + A2’B1B2

3. A = B - A1’A2’B1’B2’ + A1’A2B1’B2 + A1A2’B1B2’ + A1A2B1B2

구한 Boolean function을 이용하여 다음과 같이 Verilog 코드를 작성했다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

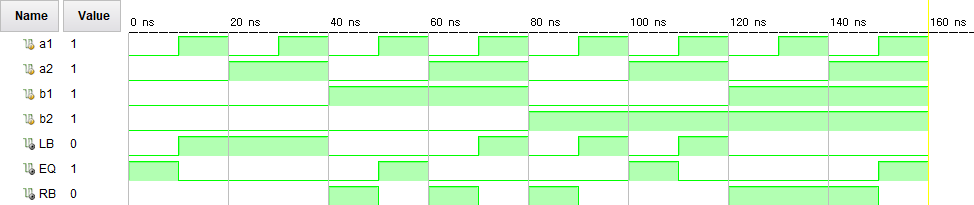
자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

각 경우에 대한 Boolean function을 그대로 디자인 코드에 작성했다. 가독성을 위해 연산자 우선순위를 생각하지 않고 ‘()’를 통해 묶어주었다. 또한 총 4개의 입력이 각 각 10,20,40,80ms마다 전환되도록 테스트 밴치코드를 구현하여 총 160ms 내에 모든 입력 경우의 수를 확인할 수 있다. 시뮬레이션 결과와 진리표는 아래와 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In A (A1)** | **In B (A2)** | **In C (B1)** | **In D (B2)** | **Out F1** | **Out F2** | **Out F3** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |



진리표 상으로 F1이 LB (A>B), F2이 EQ (A=B), F3이 RB (A<B)를 의미한다. 진리표와 시뮬레이션 결과를 비교했을 때 같은 결과가 나온다는 것을 확인할 수 있다.

**4.**

**4-1) Even & Odd Parity Bit Generator**

카르노맵을 통한 Boolean function 도출 결과 even parity bit은 모든 입력값을 XOR한 결과이고 odd parity bit은 모든 입력값을 XNOR한 결과임을 알 수 있다. 이는 논리적으로 보면 당연하게도 진리표를 생성했을 때, parity bit이 생성되는 경우가 even과 odd가 완전히 반대임을 알 수 있다. 따라서 두 식은 서로를 부정한 형태로 생성된다.

**4-2) Even & Odd Parity Bit Checker**

Parity Bit Checker역시 generator와 마찬가지로, 카르노맵을 통한 Boolean function 도출 결과 even parity check bit은 모든 입력값을 XOR한 결과이고 odd parity check bit은 모든 입력값을 XNOR한 결과임을 알 수 있다. 이도 당연히 진리표를 생성했을 때, parity check bit이 생성되는 경우가 even과 odd가 완전히 반대이기 때문에 발생한다. 역시 두 식은 서로를 부정한 형태로 생성된다.

**4-3) 2 Bit Comparator**

2-bit Comparator도 위와 동일하게 카르노맵을 통해 F1, F2, F3의 식을 각각 나타낸 후에, 식을 정리하여 베릴로그 코드를 작성하였다. 그 결과 Simulation의 결과와 진리표가 일치함을 확인할 수 있다.

**5.**

**5-1) 2D Parity Bit Checker**

실험을 통해 알아본 1차원 Parity 검사와 달리, 2차원 Parity 검사도 존재한다. 2차원 Parity 검사는 7주차 예비보고서에서도 설명했듯이 데이터를 배열의 형태로 저장하고 각 행과 열에 대한 Parity Bit을 모두 구해야 한다. 예를 들면 Even Parity 라고 가정할 경우, 다음과 같은 형태로 나타난다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |  |

위와 같이 행과 열에 대한 Parity Bit을 모두 구하여 빨간색, 파란색 부분으로 나타냈다. 이와 같이 만들어진 데이터를 전송한 후, 전송받은 쪽에서는 1차원 Parity 검사와 마찬가지로 오류가 있는지에 대한 검사를한다. 이 때, 2차원 parity bit checker은 1차원보다는 더 많은 전송오류를 더 쉽게 검출할 수 있다. 하지만 여전히 서로 다른 데이터 단위에서 서로 같은 위치에서의 두 비트가 손상되어 전송되는 경우에 대한 오류는 확인이 불가능하다는 한계가 존재한다.

**5-2) Checksum**

Parity Bit Checker는 단순히 1의 개수에 대한 짝홀수 여부를 통해 오류가 발생했는지를 확인했 다면 Checksum은 1의 발생 개수를 세는 대신 모든 Bit의 숫자를 더해 그 연산값이 홀수인지 짝수인지를 확인하는 방식이다.

Parity check의 경우 전체 Bit 중 두 개이상의 bit에서 오류가 발생하여 1의 홀, 짝 여부가 유지된다면 오류를 검출할 수 없다는 한계가 존재하지만, checksum은 전체의 합을 알고 있기 때문에 2개 이상의 bit 오류가 발생하더라도 이를 검출할 수 있다는 장점이 있다.